AD-A062 471 CALIFORNIA UNIV BERKELEY OPERATIONS RESEARCH CENTER F/G 9/2
SCHEDULING TASKS WITH EXPONENTIAL SERVICE TIMES ON NONIDENTICAL--ETC(U)
AUG 78 G WEISS, M PINEDO N00014-77-C-0299
UNCLASSIFIED ORC-78-16 NL

| OF |
AD
A062471

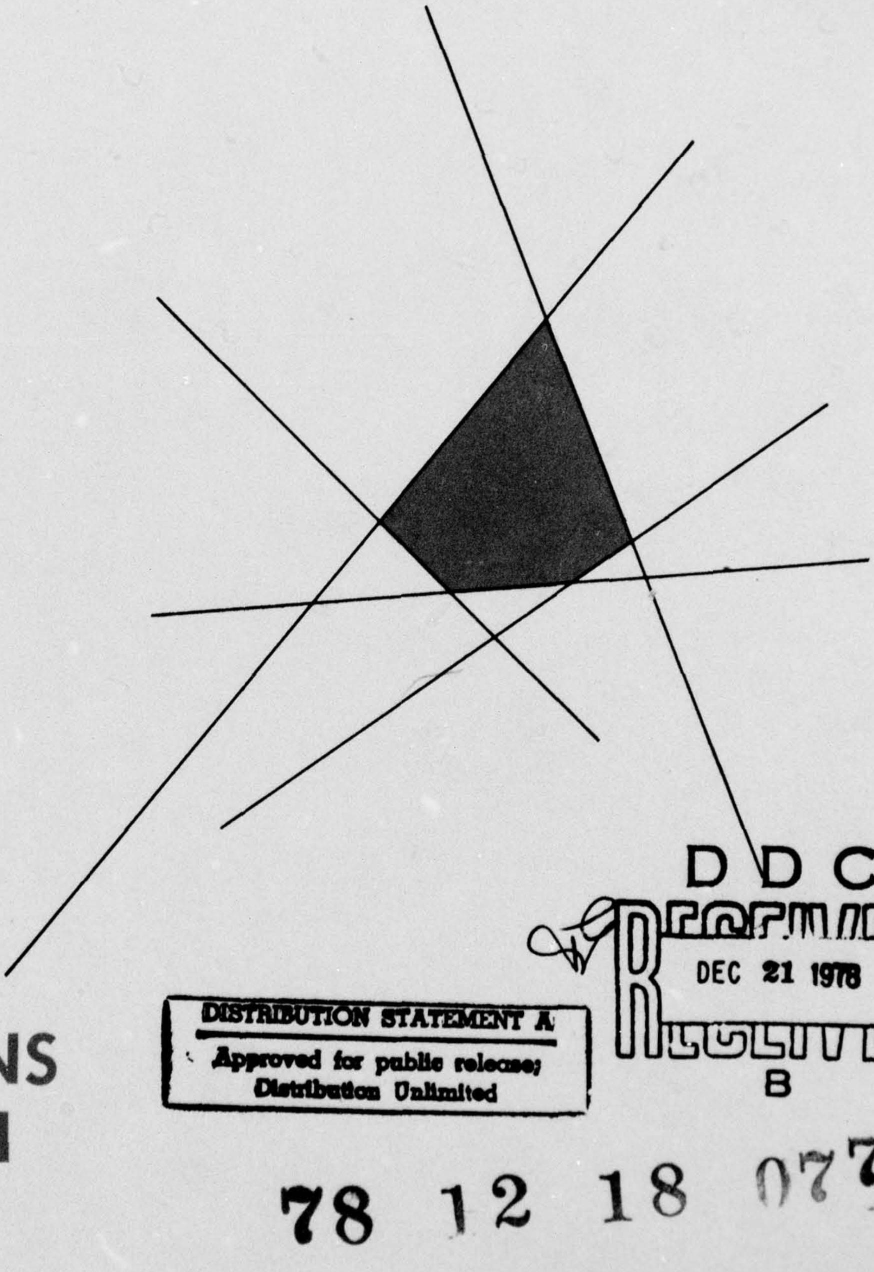
END
DATE
FILMED
3-79
DDC

ORC 78-16
AUGUST 1978

2

SCHEDULING TASKS WITH EXPONENTIAL SERVICE TIMES ON NONIDENTICAL PROCESSORS TO MINIMIZE VARIOUS COST FUNCTIONS

by
GIDEON WEISS
and
MICHAEL PINEDO



OPERATIONS
RESEARCH
CENTER

UNIVERSITY OF CALIFORNIA · BERKELEY

SCHEDULING TASKS WITH EXPONENTIAL SERVICE TIMES ON NONIDENTICAL PROCESSORS TO MINIMIZE VARIOUS COST FUNCTIONS[†]

by

Gideon Weiss
Department of Statistics
Tel-Aviv University
Tel-Aviv, Israel

and

Michael Pinedo
Instituto Venezolano de Investigaciones Cientificas
Caracus, Venezuela

AUGUST 1978

ORC 78-16

[†]This research has been partially supported by the Office of Naval Research under Contract N00014-77-C-0299 and the Air Force Office of Scientific Research (AFSC), USAF, under Grant AFOSR-77-3213 while the authors were at the University of California, Berkeley, at the Department of Statistics and the Department of Industrial Engineering and Operations Research, respectively. Reproduction in whole or in part is permitted for any purpose of the United States Government.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER (14) ORC-78-16 ; 2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER (9)
4. TITLE (and Subtitle) (6) SCHEDULING TASKS WITH EXPONENTIAL SERVICE TIMES ON NONIDENTICAL PROCESSORS TO MINIMIZE VARIOUS COST FUNCTIONS.	5. TYPE OF REPORT & PERIOD COVERED Research Report.
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) (10) Gideon/Weiss ~~and~~ Michael/Pinedo	8. CONTRACT OR GRANT NUMBER(s) (15) N00014-77-C-0299, AFOSR-77-3213
9. PERFORMING ORGANIZATION NAME AND ADDRESS Operations Research Center University of California Berkeley, California 94720	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS NR 042 379
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research Department of the Navy Arlington, Virginia 22217	12. REPORT DATE (11) August 1978
	13. NUMBER OF PAGES 33
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) (12) 34p.	15. SECURITY CLASS. (of this report) Unclassified
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

Also supported by the Air Force Office of Scientific Research (AFSC), USAF, under Grant AFOSR-77-3213.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Exponential Distribution
Dynamic Programming
Multiprocessors
Stochastic Pre-emptive Scheduling
Reliability of Series System with Spares

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

(SEE ABSTRACT)

270 750 [illegible]

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ABSTRACT

We consider preemptive scheduling of N tasks on m processors; processors have different speeds, tasks require amounts of work which are exponentially distributed, *with different parameters*. The policies of assigning at every moment the task with shortest (longest) expected processing time among those not yet completed to the fastest processor available, 2nd shortest (longest) to the 2nd fastest etc., are examined, and shown to minimize expected values of various cost functions. As special cases we obtain policies which minimize expected flowtime, expected makespan and expected lifetime of a series system with m component locations and N spares.

SCHEDULING TASKS WITH EXPONENTIAL SERVICE TIMES ON NONIDENTICAL PROCESSORS TO MINIMIZE VARIOUS COST FUNCTIONS

by

Gideon Weiss and Michael Pinedo

1. INTRODUCTION

Consider the following situation: processors $1, \ldots, m$ are available to process tasks $1, \ldots, N$. Task j needs an amount X_j of processing, where X_j is a random variable with an exponential distribution, with parameter λ_j. The processing can be done by any of the processors, and preemptions and switches of processors are allowed. Processor i has speed s_i. The processing of task j requires for completion a total time of $t_1, \ldots, t_m$ on processors $1, \ldots, m$ respectively, so that $t_1 s_1 + \cdots + t_m s_m = X_j$. $X_1, \ldots, X_n$ are assumed to be independent of each other, and independent of the manner in which processors are assigned to tasks. Starting at time 0, a cost per unit time $g\{U\}$ is incurred at time t, where $\{U\} \subset \{1, \ldots, N\}$ is the set of uncompleted tasks at t. Thus for every scheduling rule (which determines for every $\{U\} \subset \{1, \ldots, n\}$ of uncompleted tasks at t, which of the tasks should be processed on which processor) there is an expected cost over $[0,\infty)$. We want to find a scheduling rule, which minimizes the expected cost.

Let $\{U\} = \{j_1, \ldots, j_k\}$ where $\lambda_{j_1} > \lambda_{j_2} > \cdots > \lambda_{j_k}$. We consider two rules in particular: SEPT requires task j_i to be processed on processor i, for $i \leq \min(k,m)$; LEPT requires task j_{k-i+1} on processor i for $i \leq \min(k,m)$. In other words, both rules use only the fastest processors, and process on them tasks ordered by their expected length; SEPT assigns the task with the Shortest Expected Processing Time to the fastest processor, LEPT assigns the task with the Longest Expected Processing Time to the fastest processor. We find for a variety of cost rates g that SEPT or LEPT minimizes the expected total cost.

In particular we find SEPT is optimal for the following scheduling problems: to minimize the sum of completion times (flow time); to minimize the weighted sum of completion times where the weight of the completion of task j is w_j, and whenever $\lambda_j > \lambda_k$, we have $w_j > w_k$; and to minimize the sum of the first k completion times. LEPT is optimal for these scheduling problems: to minimize the last completion time (makespan); and to minimize the total amount of processing (times multipiied by speeds) done by processors $1, \ldots, r$, $(r \leq m)$, when it is assumed that for any $k \leq m$ processors $1, \ldots, k$ are used as long as $|U| \geq k$.

The situation described above can also be interpreted as a reliability problem, assuming a system with m component locations, and N spares. Component location i causes the spare to wear out at rate s_i. When spare j operates in locations $1, \ldots, m$ for total times $t_1, \ldots, t_m$, before it fails, then $X_j = t_1 s_1 + \cdots + t_m s_m$, and X_j has exponential distribution with rate λ_j. We show that if the system is a series system, then putting the best component (smallest λ) in the location with highest wearout, the second best in second highest wearout location, etc., (the equivalent of LEPT) maximizes the expected system lifetime.

The present paper generalizes results of [2], [3], [5], which deal only with identical processors, and with the minimization of flowtime and of makespan. Bruno and Downey [2] seem to have the earliest results, showing that SEPT and LEPT minimize flowtime and makespan for two $(m = 2)$ identical processors. Frederickson [3] generalizes these results to any m identical processors. In the present paper we follow some of the derivations used by Frederickson, but we use a more streamlined dynamic programming formulation which dispenses with some of the exchange arguments used by Frederickson and by Bruno and Downey. A slightly different dynamic programming formulation is used by Van der Heyde [5] to show that LEPT minimizes makespan for m identical processors. Pinedo and Weiss [4] use a different method to prove that LEPT minimizes makespan for $m = 2$ identical processors and also discuss a case in which the amount of processing has a hyperexponential distribution. Eugene L. Lawler (private communication) has suggested the investigation of nonidentical processors, and Sheldon M. Ross has suggested the reliability problem.

The paper is organized as follows: Section 2 contains a formulation of the problem as a dynamic programming problem and a derivation of a necessary and of a sufficient condition for the optimality of a priority policy. Section 3 and 4 contain sufficient conditions on the cost rate g under which SEPT or LEPT are optimal. Section 5 lists the applications of Section 3 and 4, and some counterexamples. A discussion of the sufficient conditions, and of some open problems is contained in Section 6.

2. DYNAMIC PROGRAMMING FORMULATION AND CONDITIONS OF OPTIMALITY

At time $t = 0$ we have tasks $\{1, \ldots, N\}$ and processors $\{1, \ldots, m\}$. We order the processors to have $s_1 \geq s_2 \geq \cdots \geq s_m$. We shall assume throughout that $m \geq N$. Most practical situations can be reformulated so as to have $m \geq N$ by adding processors with speeds $s = 0$. Tasks $\{1, \ldots, N\}$ are then assigned to processors, and processed until a task is completed, and a new assignment of processors is chosen. We consider $t = 0$ and the times at which tasks are completed as decision moments. The state U at a decision moment is the set of uncompleted tasks, $U \subseteq \{1, \ldots, N\}$. The set of actions available in state U, $J(U)$ is defined as:

$$J(U) = \{f \mid f \text{ is a 1-1 function from } U \text{ into } \{1, \ldots, m\}\} .$$

For state U and action f on $J(U)$, the next decision moment occurs time T later, where T is an exponential random variable with rate $\Lambda_f(U) = \sum_{j \varepsilon U} \lambda_j s_{f(j)}$. The state at the next decision moment is U' where:

$$P(U' = U - \{k\}) = \frac{\lambda_k s_{f(k)}}{\Lambda_f(U)} \qquad k \varepsilon U .$$

The cost incurred between the two decision moments is $g(U) \cdot T$, where g is a set function, with $g(\phi) = 0$.

Note that at decision moment t and state U, the effect of the choice of action f does not depend on the state at τ, $\tau \varepsilon [0,t)$. This corresponds to the situation described in the introduction, because of the memoryless property of the exponential distribution.

At each decision moment t, when the state is U, a policy π specifies which action from $J(U)$ should be taken. We use $G_\pi(U,t)$ to denote the expected value of the cost incurred over (t,∞), starting with state U at the decision moment t and using policy π. We wish to find π^* which minimizes $G_\pi(\{1, \ldots, N\},0)$. At time t and state U a stationary policy will choose an action from $J(U)$ independent of t. For such policies $G_\pi(U,t) = G_\pi(U,0)$; we will use $G_\pi(U)$ to denote $G_\pi(U,0)$. The calculation of the expected cost $G_\pi(U)$, when π chooses action f at U can be done using the recursion relation:

$$(1) \qquad G_\pi(U) = \frac{g(U) + \sum_{j\varepsilon U} \lambda_j s_{f(j)} G_\pi(U - \{j\})}{\Lambda_f(U)} .$$

We call an action $f \varepsilon J(U)$ fast if for $|U| = n$ f is 1-1 from U to $1, \ldots, n$. A policy π is fast if it uses only fast actions (i.e., since $s_1 \geq s_2 \geq \cdots \geq s_m$ only the fastest processors are used). We will call a policy a priority policy if for some particular renumbering of the tasks as $\{1, \ldots, N\}$, for every state $U = \{i_1, \ldots, i_n\}$ where $i_1 < \cdots < i_n$, π takes the action $f(i_j) = j$, $j = 1, \ldots, n$.

Let π be a priority policy. Let U be an arbitrary fixed subset of $\{1, \ldots, N\}$. We shall make the following simplifications in notation: we shall renumber the tasks in U, in order of their priorities according to π, as $U = \{1, \ldots, n\}$. Thus the action f chosen by π at U will be $f(j) = j$ for $j = 1, \ldots, n$. We shall also use the following notations: let $\Lambda = \Lambda_f(U)$, $G = G_\pi(U)$, $G_k = G_\pi(U - \{k\})$, $\Lambda_k = \sum_{j=1}^{k-1} \lambda_j s_j + \sum_{j=k+1}^{n} \lambda_j s_{j-1} = \Lambda_f(U - \{k\})$, $g = g(U)$, $g_k = g(U - \{k\})$, and $G_{k\ell} = G_\pi(U - \{k,\ell\})$. The recursion (1) in this notation gives:

(2) $$G = \frac{g + \sum_{j=1}^{n} \lambda_j s_j G_j}{\Lambda} .$$

(3) $$G_k = \frac{g_k + \sum_{j=1}^{k-1} \lambda_j s_j G_{jk} + \sum_{j=k+1}^{n} \lambda_j s_{j-1} G_{jk}}{\Lambda_k} .$$

So far we have discussed policies which allow decisions only at $t = 0$ and at task completion times; we denote these by $\mathbb{P}_o$. We could also consider a much wider class of policies, $\mathbb{P}$, which allows decisions at any time $t \in [0,\infty)$. The action space for these policies, at each t , and in the state U , would still be $J(U)$. We note then that $\mathbb{P}_o$ includes all the stationary policies in $\mathbb{P}$. We will not consider $\mathbb{P}$, because of difficulties in defining the process under general $\pi \in \mathbb{P}$. Instead we will consider for a fixed Δ the class $\mathbb{P}_\Delta$ of policies which allow decisions at $0, \Delta, 2\Delta, \ldots,$ and at task completions. In $\mathbb{P}_\Delta$ by [1] there exist optimal policies, there exists a stationary optimal policy, and a necessary and sufficient condition for π to be optimal is that π is excessive. If π is stationary (that is $\pi \varepsilon \mathbb{P}_o$) it will then be optimal in $\mathbb{P}_o$.

We say that policy π is excessive in $\mathbb{P}_\Delta$ if for any decision moment t and any state U , we can show that for all $f \varepsilon J(U)$, $G_{f|\pi}(U,t) \geq G_\pi(U,t)$. Here $G_{f|\pi}(U,t)$ is the expected cost of taking action f at t , and continuing with policy π from the next decision onwards.

To check that a stationary policy $\pi \varepsilon \mathbb{P}_o$ is optimal among all of $\mathbb{P}_o$ it is therefore sufficient to check that there exists Δ such that for every $\delta < \Delta$, every state U , and every $f \varepsilon J(U)$, $G_{f|\pi}(U) > G_\pi(U)$ where action f is taken from 0 to δ or the first completion, which ever comes first, and π is used from then on.

We derive now a condition for the optimality of a priority policy π .

<u>Theorem 1</u>:

Let π be a priority policy. Assume $s_1 > s_2 > \cdots > s_m > 0$, $m > N$.

(i) π is a unique optimal policy if for every arbitrary fixed U :

(4) $$G - G_k > 0 \qquad k = 1, \ldots, n .$$

(5) $$\lambda_k(G - G_k) - \lambda_\ell(G - G_\ell) > 0 \qquad k < \ell .$$

(ii) Conditions (4), (5) as weak inequalities are necessary for the optimality of π .

Proof:

(i) We want to show that there exists Δ such that if $\delta < \Delta$, for any action $f' \in J(U)$ if f' is taken from 0 to the minimum of δ and the first completion, and π is used from then on, then (4) and (5) imply $G_{f'|\pi}(U) > G$. This will imply strong excessivity of π in $\mathbb{P}_\Delta$, and hence unique optimality in $\mathbb{P}_\Delta$ and therefore in $\mathbb{P}_o$. For an arbitrary action f' in $J(U)$, we can write

(6) $$G_{f'|\pi} = \delta \cdot g + \sum \delta\lambda_j s_{f'(j)} G_j + (1 - \sum \delta\lambda_j s_{f(j)})G + o(\delta) .$$

The term $o(\delta)$ includes costs incurred over $(0,\delta)$ when there is an event in $(0,\delta)$, costs incurred over (δ,∞) if there is more than 1 event in $(0,\delta)$, and correction terms for the probabilities that no event occurs in $(0,\delta)$ or that task j is completed in $(0,\delta)$.

Assume $f' \neq f$. We will look at two cases:

Case 1:

There exists $k < n$ such that $f'(j) \neq k$ for any j. In that case there exists i such that $f'(i) = \ell > n$. We have $s_k > s_\ell$. Define

$$f''(j) = \begin{cases} f'(j) & j \neq i \\ k & j = i . \end{cases}$$

Case 2:

$\{f(1), \ldots, f(n)\} = \{1, \ldots, n\}$. In this case there exists $1 \le k < \ell \le n$ such that $f'(k) > f'(\ell)$. Let $s_{f'(k)} = s'$, $s_{f'(\ell)} = s''$, where $s'' > s'$. Define:

$$f''(j) = \begin{cases} f'(j) & j \neq k,\ell \\ f'(k) & j = \ell \\ f'(\ell) & j = k \ . \end{cases}$$

We use (6) to obtain, in Case 1

$$G_{f'|\pi} - G_{f''|\pi} = \delta\lambda_i(s_k - s_\ell)(G - G_i) + o(\delta) \tag{7}$$

and in Case 2

$$G_{f'|\pi} - G_{f''|\pi} = \delta(s'' - s')(\lambda_k(G - G_k) - \lambda_\ell(G - G_\ell)) + o(\delta) \ . \tag{8}$$

By (4) and (5) and by $s_k > s_\ell$, $s'' > s'$, we then have:

$$G_{f'|\pi} - G_{f''|\pi} = \delta K + o(\delta) \ , \ K > 0 \ .$$

By repeating this argument for f'' and obtaining f''' etc., we get in a finite number of steps action f . So we have:

$$G_{f'|\pi} - G_\pi = \delta K_1 + o(\delta) \qquad K_1 > 0$$

and we can now find $\Delta_{f'}$ such that for all $\delta < \Delta_{f'}$, $G_{f'|\pi} - G_\pi > 0$. Repeating for all actions in $J(U)$ (which is finite) we obtain $\Delta = \min_{f_1 \varepsilon J(U)} \Delta_{f_1}$ for which $G_{f_1|\pi} - G_\pi > 0$, for all $\delta < \Delta$, $f_1 \ \varepsilon \ J(U)$.

(ii) Assume now, in contradiction to the weak inequalities (4), (5) that:

Case 1: $G - G_k < 0$ for some k or:

Case 2: $\lambda_k(G - G_k) - \lambda_\ell(G - G_\ell) < 0$ for some $k < \ell$.

We can define in Case 1:

$$f'(j) = \begin{cases} j & j \neq k \\ m & j = k \end{cases} \qquad (m > N \geq n)$$

and in Case 2:

$$f'(j) = \begin{cases} j & j \neq k,\ell \\ k & j = \ell \\ \ell & j = k \end{cases}$$

and use (7) or (8) to show $G_{f'|\pi}(U) - G < 0$ for δ small enough, which contradicts the optimality.■

3. OPTIMALITY OF SEPT POLICIES

Let the tasks be ordered by $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_N$, and let π be the priority policy for that ordering; π is a SEPT policy.

Lemma 1:

Let π be a priority policy. Assume $g(\phi) = 0$ and $g(U) \geq g(V)$ if $U \supseteq V$. Then, for arbitrary fixed u (with the above simplified notation),

(9) $$G - G_k \geq 0 \qquad k = 1, \ldots, n ,$$

and $g(U) > g(V)$ if $U \supset V$ implies:

(10) $$G - G_k > 0 \qquad k = 1, \ldots, n .$$

Proof:

The proof is by induction on n, the size of U.

Induction Base, $n = 2$:

For $n = 2$, we have:

$$G = \frac{g\{1,2\} + \lambda_1 s_1 G_1 + \lambda_2 s_2 G_2}{\lambda_1 s_1 + \lambda_2 s_2}$$

$$G_1 = \frac{g\{2\}}{\lambda_2 s_1}$$

$$G_2 = \frac{g\{1\}}{\lambda_1 s_1} .$$

So

$$G - G_1 = \left(g\{1,2\} - \frac{s_2}{s_1} g\{2\} + \frac{\lambda_2 s_2}{\lambda_1 s_1} g\{1\} \right) / (\lambda_1 s_1 + \lambda_2 s_2) \geq$$

$$\geq (g\{1,2\} - g\{2\}) / (\lambda_1 s_1 + \lambda_2 s_2) > 0$$

and:

$$G - G_2 = \left(g\{1,2\} + \frac{\lambda_1}{\lambda_2} g\{2\} - g\{1\}\right) / (\lambda_1 s_1 + \lambda_2 s_2) \geq$$

$$\geq (g\{1,2\} - g\{1\}) / (\lambda_1 s_1 + \lambda_2 s_2) > 0 \ .$$

<u>General n</u> :

We will show the induction argument which proves (10). The proof for (9) is similar. We shall assume that (10) holds for sets of size $\leq n - 1$. Write:

$$(11) \qquad G = \frac{g}{\Lambda} + \frac{\lambda_k s_k}{\Lambda} G_k + \sum_{j \neq k} \frac{\lambda_j s_j}{\Lambda} G_j$$

$$(12) \qquad G_k = \frac{g_k}{\Lambda} + \frac{\Lambda - \Lambda_k}{\Lambda} G_k + \sum_{j=1}^{k-1} \frac{\lambda_j s_j}{\Lambda} G_{jk} + \sum_{j=k+1}^{n} \frac{\lambda_j s_{j-1}}{\Lambda} G_{jk} \ .$$

So $\left(\text{noting } \lambda_k s_k + \Lambda_k - \Lambda = \sum_{j=k+1}^{n} \lambda_j (s_{j-1} - s_j)\right)$:

$$G - G_k = \frac{g - g_k}{\Lambda} + \sum_{j \neq k} \frac{\lambda_j s_j}{\Lambda} (G_j - G_{jk}) + \sum_{j=k+1}^{n} \frac{\lambda_j (s_{j-1} - s_j)}{\Lambda} (G_k - G_{jk}) > 0$$

since $g > g_k$, $s_{j-1} \geq s_j$ and $G_j > G_{jk}$, $G_k > G_{jk}$ by the induction assumption. ■

We now prove

<u>Theorem 2</u>:

The SEPT policy π is optimal if g satisfies, for every $U \subseteq \{1, \dots, N\}$

(13) $$g(\phi) = 0\ ,\ g(U) \geq 0\ .$$

(14) $$g(U - \{\ell\}) \geq g(U - \{k\}) \qquad k < \ell\ ,\ k\ ,\ \ell \in U\ .$$

(15) $$g(U) - g(U - \{k\}) - g(U - \{\ell\}) + g(U - \{k,\ell\}) \geq 0 \qquad k\ ,\ \ell \in U\ .$$

Proof:

We note first by (13) and (15) that

(16) $$G(U) \geq G(V) \quad \text{for} \quad U \supseteq V\ .$$

Assume first that $\lambda_1 > \lambda_2 > \cdots > \lambda_N > 0$, $s_1 > s_2 > \cdots > s_m > 0$, $m > N$, and that (13) - (15) and hence (16) hold as strict inequalities. We then show that π is the unique optimal policy.

Let U be an arbitrary fixed subset of $\{1, \ldots, N\}$ then we know from the preceding lemma that (10) holds. Using the above simplified notation we now will show:

(17) $$G - G_k - G_\ell + G_{k\ell} > 0\ .$$

(18) $$\lambda_{k-1}(G - G_{k-1}) - \lambda_k(G - G_k) > 0\ .$$

The proof is by induction on n , the size of U .

Induction Base, $n = 2$:

For $n = 2$ we have (noting that $G_{12} = 0$) :

$$G - G_1 - G_2 = \frac{g\{1,2\} - g\{1\} - \frac{s_2}{s_1} g\{2\}}{\lambda_1 s_1 + \lambda_2 s_2} > 0$$

which proves (17) and

$$\lambda_1(G - G_1) - \lambda_2(G - G_2) = \frac{(\lambda_1 - \lambda_2)s_1 g\{1,2\} - (s_1 + s_2)(\lambda_1 g\{2\} - \lambda_2 g\{1\})}{(\lambda_1 s_1 + \lambda_2 s_2)s_1} >$$

$$(19) \qquad \frac{(\lambda_1 s_1 + \lambda_2 s_2)g\{1\} - (\lambda_2 s_1 + \lambda_1 s_2)g\{2\}}{(\lambda_1 s_1 + \lambda_2 s_2)s_1} >$$

$$\frac{(\lambda_1 - \lambda_2)(s_1 - s_2)g\{2\}}{(\lambda_1 s_1 + \lambda_2 s_2)s_1} > 0$$

which proves (18).

General n :

We shall assume (17) and (18) hold for sets of size $\leq n - 1$. To prove (17) write:

$$(20) \qquad G_\ell = \frac{g_\ell}{\Lambda} + \frac{\Lambda - \Lambda_\ell}{\Lambda} G_\ell + \sum_{j=1}^{\ell-1} \frac{\lambda_j s_j}{\Lambda} G_{j\ell} + \sum_{j=\ell+1}^{n} \frac{\lambda_j s_{j-1}}{\Lambda} G_{j\ell}$$

$$(21) \qquad G_{k\ell} = \frac{g_{k\ell}}{\Lambda} + \frac{\Lambda - \Lambda_{k\ell}}{\Lambda} G_{k\ell} + \sum_{j=1}^{k-1} \frac{\lambda_j s_j}{\Lambda} G_{jk\ell} + \sum_{j=k+1}^{\ell-1} \frac{\lambda_j s_{j-1}}{\Lambda} G_{jk\ell} + \sum_{j=\ell+1}^{n} \frac{\lambda_j s_{j-2}}{\Lambda} G_{jk\ell} .$$

We have, substituting (11), (12), (20), (21):

$$\Lambda(G - G_k - G_\ell + G_{k\ell}) = (g - g_k - g_\ell + g_{k\ell})$$

$$+ \sum_{j\neq k,\ell} \lambda_j s_j (G_j - G_{jk} - G_{j\ell} + G_{jk\ell})$$

$$+ \sum_{j=k+1}^{n-1} \lambda_j (s_{j-1} - s_j)(G_k - G_{jk} - G_{k\ell} + G_{jk\ell})$$

(22)

$$+ \sum_{j=\ell+1}^{n} \lambda_j (s_{j-1} - s_j)(G_\ell - G_{j\ell} - G_{k\ell} + G_{jk\ell})$$

$$+ \sum_{j=\ell}^{n-1} (s_{j-1} - s_j)[\lambda_j (G_{k\ell} - G_{jk\ell}) - \lambda_{j+1}(G_{k\ell} - G_{j+1k\ell})]$$

$$+ \lambda_n (s_{n-1} - s_n)(G_k - G_{kn}) > 0$$

since all terms are > 0, the first by (15), the next three by (17) for $(n - 1)$, the fifth by (18) for $(n - 1)$, the last by (10).

To prove (18) write:

$$\sum_{j=1}^{k-2} \lambda_j s_j \{\lambda_{k-1}(G_j - G_{jk-1}) - \lambda_k (G_j - G_{jk})\}$$

$$+ \sum_{j=k+1}^{n} \lambda_j s_{j-1} \{\lambda_{k-1}(G_j - G_{jk-1}) - \lambda_k (G_j - G_{jk})\}$$

$$= (\lambda_{k-1} - \lambda_k)\Big\{\Lambda G - g - \lambda_{k-1} s_{k-1} G_{k-1} - \lambda_k s_k G_k$$

$$+ \sum_{j=k+1}^{n} \lambda_j (s_{j-1} - s_j) G_j\Big\} - \lambda_{k-1}\{\Lambda_{k-1} G_{k-1} - g_{k-1}$$

$$- \lambda_k s_{k-1} G_{k-1k}\} + \lambda_k \{\Lambda_k G_k - g_k - \lambda_{k-1} s_{k-1} G_{k-1k}\} .$$

Hence by rearranging and collecting terms:

$$\Lambda_k\{\lambda_{k-1}(G - G_{k-1}) - \lambda_k(G - G_k)\}$$

$$= \sum_{j=1}^{k-2} \lambda_j s_j\{\lambda_{k-1}(G_j - G_{jk-1}) - \lambda_k(G_j - G_{jk})\}$$

$$(23) \qquad + \sum_{j=k+1}^{n} \lambda_j s_{j-1}\{\lambda_{k-1}(G_j - G_{jk-1}) - \lambda_k(G_j - G_{jk})\}$$

$$+ (\lambda_{k-1} - \lambda_k)\left[\left\{g - \frac{\lambda_{k-1}}{\lambda_{k-1} - \lambda_k} g_{k-1} + \frac{\lambda_k}{\lambda_{k-1} - \lambda_k} g_k\right\}\right.$$

$$\left. - \left\{\sum_{j=k}^{n} \lambda_j s_j(G - G_j) - \sum_{j=k+1}^{n} \lambda_j s_{j-1}(G - G_j)\right\}\right] .$$

The first two terms are positive by (18) for $(n - 1)$. We still need to show:

$$g - \frac{\lambda_{k-1}}{(\lambda_{k-1} - \lambda_k)} g_{k-1} + \frac{\lambda_k}{\lambda_{k-1} - \lambda_k} g_k >$$

$$(24) \qquad > \sum_{j=k}^{n} \lambda_j s_j(G - G_j) - \sum_{j=k+1}^{n} \lambda_j s_{j-1}(G - G_j) .$$

By (14)

$$g - \frac{\lambda_{k-1}}{\lambda_{k-1} - \lambda_k} g_{k-1} + \frac{\lambda_k}{\lambda_{k-1} - \lambda_k} g_k > g - g_k$$

but:

$$g = \Lambda G - \sum_{j=1}^{n} \lambda_j s_j G_j = \sum_{j=1}^{n} \lambda_j s_j (G - G_j)$$

$$g_k = \sum_{j=1}^{k-1} \lambda_j s_j (G_k - G_{jk}) + \sum_{j=k+1}^{n} \lambda_j s_{j-1}(G_k - G_{jk}) ,$$

so:

$$g - g_k = \sum_{j=k}^{n} \lambda_j s_j (G - G_j) - \sum_{j=k+1}^{n} \lambda_j s_{j-1} (G - G_j)$$

$$+ \sum_{j=1}^{k-1} \lambda_j s_j (G - G_j - G_k + G_{jk})$$

$$+ \sum_{j=k+1}^{n} \lambda_j s_{j-1} (G - G_j - G_k + G_{jk}) ,$$

but the last two terms are positive by (17) for $(n - 1)$, which proves (24) and hence (18). By Theorem 1, (10) and (18) imply that π is uniquely optimal.

Suppose now we only have $s_1 \geq \cdots \geq s_m \geq 0$, $\lambda_1 \geq \cdots \geq \lambda_N$, and (13), (14) and (15) only hold as weak inequalities. We can perturb λ's , s's and g very slightly to get strong inequalities. For the perturbed problem, let $G^{\varepsilon}_{\pi'}(U)$ be the expected cost under stationary $\pi' \in \mathbb{P}_o$ (of which there are a finite number). Then $G^{\varepsilon}_{\pi}(U) < G^{\varepsilon}_{\pi'}(U)$ for all stationary $\pi' \in \mathbb{P}_o$. Letting the perturbation approach zero, we obtain $G_{\pi}(U) \leq G_{\pi'}(U)$, so π is optimal (possibly not unique) in $\mathbb{P}_o$.■

4. OPTIMALITY OF LEPT POLICIES

Let tasks 1, ..., N be ordered so that $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$, and let π be the priority policy for that ordering; π is a LEPT policy.

Theorem 3:

The LEPT policy π is optimal if g is of the form:

(25) $$g(U) = c_{|U|} ,$$

where:

(26) $$c_o = 0 , \; c_k \geq c_{k-1} \qquad k = 1,2, \ldots, N ,$$

and the c_k's satisfy:

(27) $$\frac{c_k - c_{k-1}}{s_k} \leq \frac{c_{k-1} - c_{k-2}}{s_{k-1}} \qquad k = 2, \ldots, N ,$$

(where we let 0/0 stand for 0).

Proof:

We shall again assume strong inequalities in (26) and (27), and that $s_1 > s_2 > \cdots > s_m > 0$, $m > N$, and $\lambda_1 < \lambda_2 < \cdots < \lambda_n$. We will show that π is uniquely optimal. A perturbation argument as in Theorem 2 then implies optimality of π for weak inequalities. We consider again an arbitrary fixed U , using the simplified notation described above.

We note that $g(U) > g(V) > g(\phi) = 0$ if $U \supset V \neq \phi$. Hence by Lemma 1, $G - G_k > 0$ for all k .

We now show that

(28) $$\lambda_{k-1}(G - G_{k-1}) - \lambda_k(G - G_k) > 0 \qquad k = 2, \ldots, n$$

by induction on n .

<u>Induction Base, $n = 2$</u> :

We obtain as in (19):

$$\lambda_1(G - G_1) - \lambda_2(G - G_2) =$$

$$\frac{(s_1 + s_2)(\lambda_2 g\{1\} - \lambda_1 g\{2\}) - (\lambda_2 - \lambda_1)s_1 g\{1,2\}}{(\lambda_1 s_1 + \lambda_2 s_2)s_1} =$$

$$\frac{(\lambda_2 - \lambda_1)\{(s_1 + s_2)c_1 - s_1 c_2\}}{(\lambda_1 s_1 + \lambda_2 s_2)s_1}$$

but, by (27), $\frac{c_2 - c_1}{s_2} < \frac{c_1 - c_0}{s_1}$, $c_0 = 0$, so:

$$s_1 c_2 < (s_1 + s_2)c_1$$

and (28) follows.

<u>General n</u> :

We assume (28) holds for sets of size $(n - 1)$. We rewrite (23), noting that $\lambda_k > \lambda_{k-1}$, $g - \frac{\lambda_{k-1}}{\lambda_{k-1} - \lambda_k} g_{k-1} + \frac{\lambda_k}{\lambda_{k-1} - \lambda_k} g_k = c_n - c_{n-1}$, and rearranging the last term, as:

$$\Lambda_k\{\lambda_{k-1}(G - G_{k-1}) - \lambda_k(G - G_k) =$$

$$\sum_{j=1}^{k-2} \lambda_j s_j\{\lambda_{k-1}(G_j - G_{jk-1}) - \lambda_k(G_j - G_{jk}\} +$$

(29) $$\sum_{j=k+1}^{n} \lambda_j s_{j-1}\{\lambda_{k-1}(G_j - G_{jk-1}) - \lambda_k(G_j - G_{jk})\} +$$

$$(\lambda_k - \lambda_{k-1})\left[\sum_{j=k+1}^{n} s_{j-1}\{\lambda_{j-1}(G - G_{j-1}) - \lambda_j(G - G_j)\} + \lambda_n s_n(G - G_n) - (c_n - c_{n-1})\right] .$$

The first two summations are positive by the induction hypothesis. We shall show that

(30) $$\lambda_n s_n(G - G_n) > (c_n - c_{n-1}) ,$$

which will be sufficient to prove (28) for $k = n$. Assuming then (as an inner induction hypothesis) that (28) holds for $n, n - 1, \ldots, k + 1$, the third summation will be positive, and so (28) will follow for k .

To show (30) we calculate $G - G_n$. We note that because π is a priority policy, the completion times of tasks $1, \ldots, n - 1$ are the same whether we start with tasks $1, \ldots, n$ or with tasks $1, \ldots, n - 1$ at $t = 0$. Let $V(t)$, $t \in [0,\infty)$, be the subset of $\{1, \ldots, n - 1\}$ of uncompleted tasks at time t , when $V(0) = \{1, \ldots, n - 1\}$. $G - G_n$ is then the expected value of $\int_0^\infty [g(V(t) \cup \{n\}) - g(V(t))]dt$. Assume that tasks $1, \ldots, n - 1$ [illegible] pleted in the order $i_1, \ldots, i_{n-1}$, at times $0 < t$ [illegible] $_2 < \cdots < t_{n-1}$, $t_o = 0$. Then:

$$G - G_n = \frac{g\{i_1, \ldots, i_{n-1}, n\} - g\{i_1, \ldots, i_{n-1}\}}{\lambda_n s_n}\left(1 - e^{-\lambda_n s_n t_1}\right)$$

$$+ \frac{g\{i_2, \ldots, i_{n-1} n\} - g\{i_2, \ldots, i_{n-1}\}}{\lambda_n s_{n-1}}\left(1 - e^{-\lambda_n s_{n-1}(t_2 - t_1)}\right) e^{-\lambda_n s_n t_1}$$

$$+ \cdots$$

$$(31) \quad + \frac{g\{i_{k+1}, \ldots, i_{n-1}, n\} - g\{i_{k+1}, \ldots, i_{n-1}\}}{\lambda_n s_{n-k}}\left(1 - e^{-\lambda_n s_{n-k}(t_{k+1} - t_k)}\right)$$

$$\prod_{j=0}^{k-1} e^{-\lambda_n s_{n-j}(t_{j+1} - t_j)}$$

$$+ \cdots$$

$$+ \frac{g\{n\} - g\{\phi\}}{\lambda_n s_1} \prod_{j=0}^{n-2} e^{-\lambda_n s_{n-j}(t_{j+1} - t_j)} .$$

In the case described above, $g\{i_{k+1}, \ldots, i_{n-1}, n\} - g\{i_{k+1}, \ldots, i_{n-1}\} = c_{n-k} - c_{n-k-1}$. By (27), $\frac{c_n - c_{n-1}}{s_n} < \frac{c_{n-k} - c_{n-k-1}}{s_{n-k}}$ for $k = 1, \ldots, n - 1$, so by substituting $\frac{c_n - c_{n-1}}{s_n}$ the expression would decrease. The terms multiplying $\frac{c_{n-k} - c_{n-k-1}}{\lambda_n s_{n-k}}$, $k = 0, \ldots, n - 1$ add up to 1 , and so we have

$$G - G_n > \frac{c_n - c_{n-1}}{\lambda_n s_n}$$

which proves (30) and the theorem.■

5. APPLICATIONS

In this section we list some problems that are optimized by SEPT or LEPT. In general we note that for the special case where $s_1 = \cdots = s_m$, $N > m$, SEPT and LEPT are nonpreemptive. When $s_1 > \cdots > s_m$ preemptions will be needed. Thus in the following problems, when $s_1 = \cdots = s_m$, $N > m$, we show SEPT or LEPT to be optimal in the class of nonpreemptive policies. Also, by adding enough processors with speed 0, we see that some of these will not be used by the optimal policies. Hence the SEPT and LEPT remain optimal when we allow insertions of idle time.

(1) SEPT Minimizes Expected Flow Time

We denote by $C_1, C_2, \ldots, C_N$ the completion times of jobs $1, \ldots, N$, and by $0 \le T_1 \le \cdots \le T_N$ the completion times in their order of occurrence. The expected flow time of a policy π is

$$F_\pi = E\left(\sum_{j=1}^{N} C_j\right) = E\left(\sum_{j=1}^{N} T_j\right) .$$

The cost rate that yields this is:

$$g(U) = |U| ,$$

and it is easily seen that g satisfies the conditions of Theorem 3.

(2) SEPT Minimizes the Expected Sum of the First k Completions

We define

$$G = E\left(\sum_{j=1}^{k} T_j\right) \qquad k = 1, 2, \ldots, N .$$

The cost rate which yields this is

$$g(U) = \begin{cases} |U| - (n - k) & \text{for } |U| > (n - k) \\ 0 \quad \text{for } |U| \leq (n - k) \end{cases}$$

which satisfies the conditions of Theorem 3.

(3) SEPT Minimizes the Expected Weighted Sum of Completion Times (Agreeable Weights)

Let a cost of $w_j \geq 0$ per unit time be associated with task j, that is

$$G = E\left(\sum_{j=1}^{N} w_j C_j\right) .$$

We call the costs "agreeable" if $\lambda_k > \lambda_\ell \Rightarrow w_k \geq w_\ell$. The appropriate cost rate is

$$g(U) = \sum_{j \in U} w_j .$$

The priority ordering to be used is $1, \ldots, N$, such that $\lambda_k > \lambda_{k+1}$ and if $\lambda_k = \lambda_{k+1}$, $w_k \geq w_{k+1}$, $k = 1, \ldots, N - 1$. Obviously $g(\phi) = 0$, $g(U) \geq 0$, $g(U - \{\ell\}) - g(U - \{k\}) = w_k - w_\ell \geq 0$ for $k < \ell$, and $g(U) - g(U - \{k\}) - g(U - \{\ell\}) + g(U - \{k,\ell\}) = 0$, so Theorem 3 applies. Case 1 above is a special case of this problem with $w_j = 1$, Case 2 is not a special case of this problem. A different special case is the sum of completion times of the k shortest tasks, obtained by $w_1 = \cdots = w_k = 1$, $w_{k+1} = \cdots = w_N = 0$.

(4) SEPT or the "cμ" Rule Do Not Minimize the Expected Weighted Sum of Completion Times In General

The "cμ" rule is the priority policy which arranges the tasks as $1, \ldots, N$ with $w_1\lambda_1 \geq w_2\lambda_2 \geq \cdots \geq w_N\lambda_N$. When weights are "agreeable" it coincides with SEPT. The following two counter-examples show that neither SEPT nor "cμ" are optimal in general.

(i) $n = 3$, $s_1 = s_2 = 1$, $s_3 = 0$ (identical processors)

$\lambda_1 = 3 \quad w_1 = 3$

$\lambda_2 = 3 \quad w_2 = 1$

$\lambda_3 = 1 \quad w_3 = 2$

$$G^{12} = \frac{1}{6}\left[6 + 3(1 + 2) + 3\left(\frac{1}{3} + 2\right)\right] = \frac{132}{36}$$

$$G^{13} = \frac{1}{4}\left[6 + 3\left(\frac{1}{3} + 2\right) + 1\left(1 + \frac{1}{3}\right)\right] = \frac{129}{36}$$

$$G^{23} = \frac{1}{4}\left[6 + 3(1 + 2) + 1\left(1 + \frac{1}{3}\right)\right] = \frac{147}{36}$$

so processing 1,3 first is optimal. It is neither SEPT, nor $c\mu$.

(ii) $n = 2$, $s_1 = 2 \quad s_2 = 1$

$\lambda_1 = 3 \quad w_1 = 1$

$\lambda_2 = 1 \quad w_2 = 2$

$$G^{12} = \frac{1}{7}\left[3 + 6 \times 1 + 1 \times \frac{1}{6}\right] = \frac{275}{210}$$

$$G^{21} = \frac{1}{5}\left[3 + 3 \times 1 + 2 \times \frac{1}{6}\right] = \frac{266}{210}$$

so putting task 2 on 1 and 1 on 2 is optimal, and is not SEPT or $c\mu$.

<u>(5) LEPT Minimizes Expected Makespan</u>

The makespan is the time from 0 until all tasks are completed. The expected makespan is:

$$M = E(T_N) = E(\max(C_1, \ldots, C_N)) .$$

This is obtained by:

$$g(U) = \begin{cases} 1 & \text{for } U \neq \emptyset \\ 0 & \text{for } U = \emptyset \end{cases}$$

which satisfies the conditions of Theorem 3.

<u>(6) SEPT Does Not Maximize The Expected Makespan</u>

The following counterexample shows that SEPT does not maximize the expected makespan. Take

$$\lambda_i = 10^6 \quad \text{for} \quad 1 \leq i \leq 10^8$$

$$\lambda_{10^8+1} = 2$$

$$\lambda_{10^8+2} = 1 .$$

We have 2 identical processors and allow no preemptions. Then the schedule $1, 2, \ldots, 10^8 + 2$ gives an expected makespan of approximately $51 \frac{1}{6}$ and the schedule $10^8 + 1, 1, 2, \ldots, 10^8, 10^8 + 2$ gives an expected makespan of $51 \frac{1}{4}$. We conjecture that in the class of nonpremptive policies the following schedule is optimal: If $\lambda_1 \geq \cdots \geq \lambda_N$ there exists an i, $1 \leq i < N$, such that the schedule $i, i-1, \ldots, 2, 1, i+1, i+2, \ldots, N-1, N$ maximizes the expected makespan.

(7) Minimization and Maximization of Expected Amounts of Work

Let $M_1, \ldots, M_m$ be the total times during which processors $1, \ldots, m$ are occupied. For a fast policy $M_1 = T_N$, $M_2 = T_{N-1}$, ..., $M_m = T_{N-m+1}$ (here we assume $m \leq N$) . The amount of work done by processor k is then $s_k T_{N-k+1}$. It is desired to minimize:

$$W_r = E\left(\sum_{i=1}^{r} s_i T_{N-i+1}\right) .$$

This is obtained from

$$g(U) = \begin{cases} s_1 + \cdots + s_r & |U| \geq r \\ s_1 + \cdots + s_{|U|} & |U| < r . \end{cases}$$

It is easily seen that g satisfies the conditions of Theorem 3. Hence W_r is minimized by LEPT, for every r $(1 \leq r \leq m)$. Thus we have:

LEPT minimizes the expected amount of work done by processors $1, \ldots, r$ $(s_1M_1 + \cdots + s_rM_r)$, among all fast policies.

Also, because $E(s_1M_1 + \cdots + s_mM_m) = \frac{1}{\lambda_1} + \cdots + \frac{1}{\lambda_N}$, LEPT maximizes the expected amount of work done by processors $r+1, \ldots, m$, among all fast policies.

(8) Maximizing the Expected Life Length of a Reliability System

Given a series system of m component locations and $N > m$ spares, where the spares have exponentially distributed life lengths with parameters $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_N$ and the component locations cause wearout at rates $s_1 \geq \cdots \geq s_m$, it is desired to allocate spares to component locations so as to maximize the expected system lifetime. The problem is obviously equivalent to scheduling N tasks on m processors, where only fast policies are allowed and $E(M_m) = E(T_{N-m+1})$ is to be maximized. Thus (using Case (6) with $r = m - 1$), the expected lifetime of the system is maximized when the best remaining component (one with smallest λ) is assigned (preemptively) to location 1, the 2nd best to location 2, etc.

6. DISCUSSION

We have derived some sufficient conditions on the cost rate g, under which SEPT or LEPT minimize total expected cost. These conditions include the requirement that $g(\phi) = 0$, $g(U) \geq g(V)$ for $U \supseteq V$. This means that cost decreases as the set of uncompleted tasks decreases, and from the proof of Theorem 1 and Lemma 1 it seems that it is a necessary condition for the existence of an optimal policy which is fast (both SEPT and LEPT are fast).

The sufficient condition for SEPT to be optimal is:

$$g(U) - g(U_k) - g(U_\ell) + g(U_{k\ell}) \geq 0 \qquad k, \ell \in U .$$

In the proof of Theorem 3, this is used to show that

$$\lambda_{k-1}(G - G_{k-1}) > \lambda_k(G - G_k) , \tag{28}$$

and it seems that this is the most general condition under which (28) holds.

The sufficient condition for LEPT to be optimal is less general, and we conjecture that it is enough to require:

$$g(U_k) \leq g(U_\ell) \qquad k < \ell ,$$

and

$$\frac{g(U) - g(U_k)}{s_n} \leq \frac{g(U_k) - g(U_{k,\ell})}{s_{n-1}} \qquad k, \ell \in U .$$

One generalization of the problems discussed above is to allow arrivals of new tasks. We conjecture that LEPT remains optimal, since it minimizes makespan and thus will yield short busy periods.

Another direction to generalize the results is to look at general distributions. The crucial question then is to find the appropriate quantity by which to order the tasks. For a very special case, mixtures of two fixed exponential random variables, SEPT minimizes flow time and LEPT minimizes makespan [4].

REFERENCES

[1] Blackwell, David, "Discrete Dynamic Programming," Annals of Mathematical Statistics, Vol. 33, pp. 719-762, (1962).

[2] Bruno, John and Peter Downey, "Sequencing Tasks with Exponential Service Times on Two Machines," Technical Report, Department of Computer Sciences, University of California, Santa Barbara, (1977).

[3] Frederickson, G. N., "Sequencing Tasks with Exponential Service Times to Minimize the Expected Flow Time Or Makespan," Department of Computer Sciences, Pennsylvania State University Report CS-78-07, (1978).

[4] Pinedo, Michael and Gideon Weiss, "Scheduling Stochastic Tasks On Two Parallel Processors," (to appear).

[5] Van der Heyden, Ludo, "A Note on Scheduling Jobs with Exponential Processing Times On Identical Processors So As To Minimize Makespan," (to appear in Mathematics of Operations Research).